Middle East Technical University | Department of Computer Engineering

CENG462

Artificial Intelligence

Spring 2022-2023

Homework 1

Due: April 12th, 2023

In this homework you will model a 2D platform and the actions of an agent. You will help the agent reach its goal in the 2D platform using A* and Iterative Deepening A* algorithms.

Problem Definition

The platform is a $6 \times N$ grid and contains empty cells, walls, rocks, the exit gate, and the agent. The specifications of each is as follows:

Empty cells The agent can move to or push a rock into empty cells. The 'moved-in' cells become occupied with the new object, and the 'moved-out' cells become empty.

Walls Walls are immovable. Further, they are blocking objects, *i.e.*, nothing can pass through them. However, the agent can climb on them or lift a rock to them as detailed below in the move specifications. Also, it is possible for a rock, the agent, or the exit gate to stay on a wall.

Rocks Rocks are movable objects that can be pushed and lifted by the agent. Nonetheless, they are blocking objects like walls. Similar to the case with the walls, the agent can climb on or lift a rock to a rock. A rock, the agent, or the exit gate can stay on a rock.

Exit Gate Exit gate denotes the goal of the agent. It absorbs anything that steps on it: if a rock is pushed or falls into it, the rock will be removed from the platform. Similarly, if the agent moves into it, the agent will be removed from the platform and will be considered to have reached its goal. There is only one exit gate in each stage.

Agent The agent can move in the platform and change the position of the rocks. The set of possible moves for the agent are to move right (R), move left (L), climb right (CR), and climb left (CL).

The specifications of the moves are as follows:

Move Right (R) If the cell located to the right of the agent

i) is out of the grid or a wall, the agent stays put

ii) is empty, the agent moves there

iii) is a rock, the agent first checks if anything is on top of the rock. If so, the agent stays put. If not, the agent forces the rock to the next cell (two cells right to the agent). In this case, if the next cell

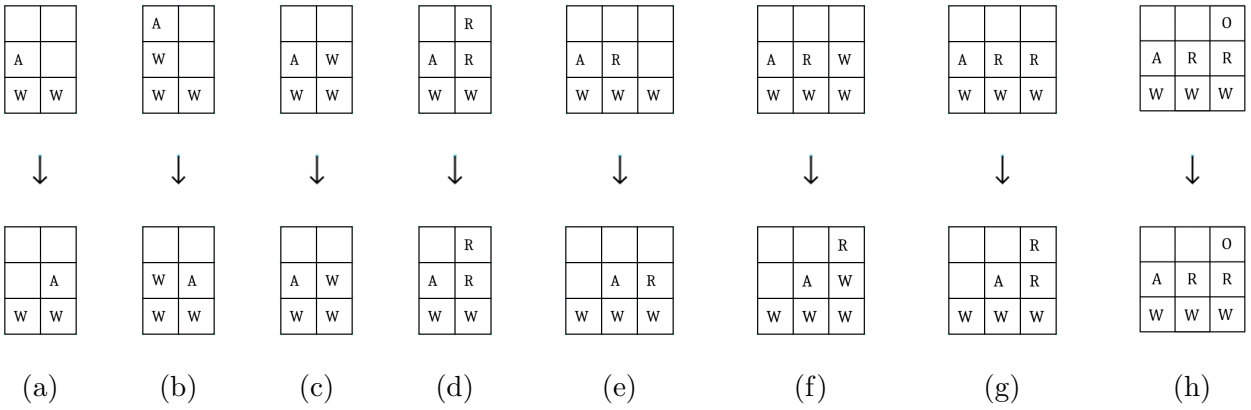
   (a) is out of grid, the agent stays put

   (b) is empty, the agent pushes the rock to the empty cell and moves to right

   (c) is a blocking object and nothing is on the blocking object, the agent lifts the rock to the top of the blocking object and moves right

See Table 1 for some sample moves. Table 1 (b) also shows the *free fall* which is explained below.

Table 1: Some cases for *Move Right* in a portion of the 2D world. W stands for a wall, R stands for a rock, A stands for the agent, O stands for any object (wall, rock, or the gate). Top row: before agent tries to move right, bottom row: after the attempt.



Move Left (L) The same specifications as Move Right applies, except replace every appearance of the word *right* with *left*.

Climb Right (CR) If the cell located to the right of the agent

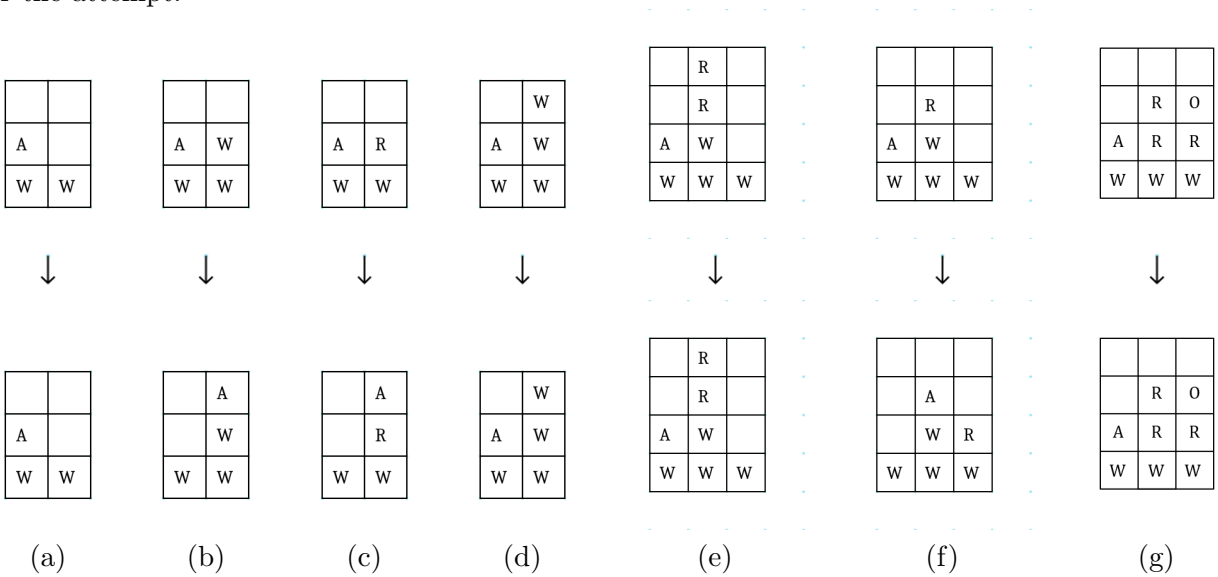
i) is empty or the exit gate, the agent stays put, as in Table 2 (a).

ii) is a blocking object, the agent checks whether the cell on top of the blocking object is empty or the exit gate. If so, the agent moves there, as in Table 2 (b)&(c). If there is a wall the agent stays put, Table 2 (d). Finally, if there is a rock:

   (a) the agent first checks if another object is on top of the rock. In this case it stays put, as in Table 2 (e).

   (b) the agent then checks if the cell that it will push the rock to is empty. If so, it will push and climb right, Table 2 (f). If not it will stay put, Table 2 (g).

Table 2: Some cases for *Climb Right*. W stands for a wall, R stands for a rock, A stands for the agent, O stands for any object (wall, rock, or the gate). Top row: before agent tries to climb right, bottom row: after the attempt.



Climb Left (CL) The same specifications as Climb Right applies, except replace every appearance of the word *right* with *left*.

Free Fall This is **not** an agent move, rather, is a mechanic of the 2D world we consider. None of the objects in the 2D world can float on empty cells. Therefore, if an object moves into a cell that is not supported by a blocking object (a wall or a rock), it falls to the top of the first blocking object underneath it. If the exit gate is on the landing cell, then the object is removed.

Examples of free fall are shown in Table 1 (b) where the agent free falls, and Table 2 (f) where the rock free falls.

Your purpose in this setting is to find the move sequences of the agent that leads him to the exit gate.

Specifications

- You are going to model the 2D world, and implement the actions of the agent, the A* and the Iterative Deepening A* algorithms.
- You will use the following function as the heuristic function:

  $$f = g + 2 \times h$$

  where g is the number of moves made by the agent and h can be computed as below:

  ```
  dist = 0
  for col=column(agent) to column(gate); do
      if height(col) > height(next(col)) - 2
          dist += 1
      else:
          dist += height(next(col)) - height(col) - 1
  done
  return dist
  ```

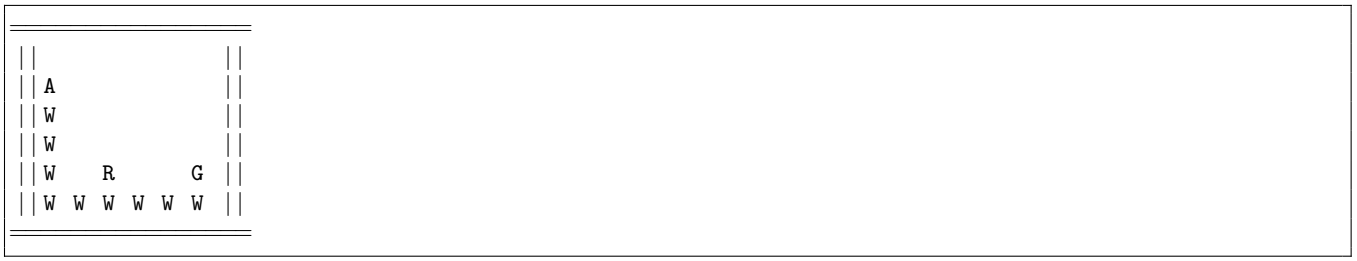
  In the above, interpret `column(.)` as returning the current column of its argument, `next(col)` as returning the adjacent column towards the gate, and interpret `height(col)` as returning the row of the first (from top to bottom) **wall** in the column `col`.
- Each task will be specified by the file name given from standard input, and the result will be printed to standard output.
  - An input reader is provided to you in `starter.py`. It is compatible with any input that will be used in grading your submissions.
  - The first line, which will either be `A*` or `IDA*`, decides on the method you will use for a given task.
  - The output should be `FAILURE` in case a solution cannot be found, or `SUCCESS` otherwise. If you print `SUCCESS`, the next line should print comma-separated moves of the agent that leads it to the exit gate.
  - The moves you print to standard output will be simulated on a separate implementation for verification. Therefore, you are free to implement any kind of tie-breaking/state ordering as you see fit.
- The file `starter.py` assumes some design choices which are not necessarily best fits for the problem. It is only aimed to ease your start to the assignment. You can rewrite the code from scratch, if you wish to do so.

Sample Input - Output

Input 1:

```
A*
W 3 0 0 0 0 0
R 2
A 0
G 5
```

Described scene:



Output 1:

```
SUCCESS
R,CR,R,R,R
```

Input 2:

```
A*
W 2 0 4 0 0 2 3
A 0
G 6
```

Described scene:

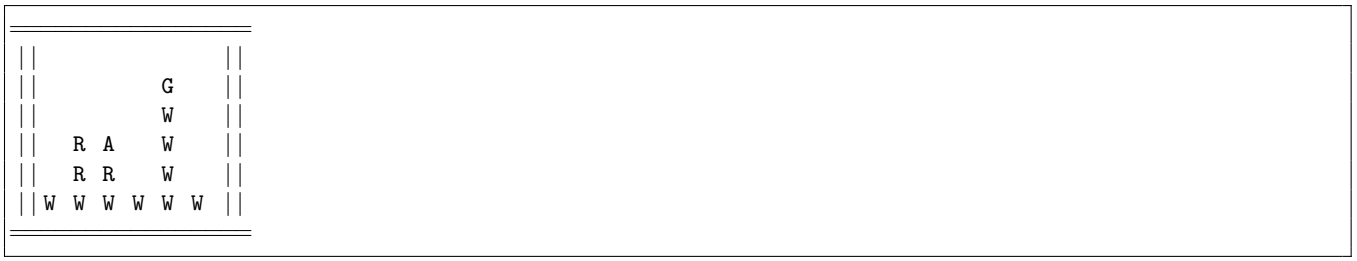
Output 2:

```
FAILURE
```

Input 3:

```
IDA*
W 0 0 0 0 3 0
R 1 1 2
A 2
G 4
```

Described scene:



Output 3:

```
SUCCESS
CL,L,CR,R,L,L,R,CR,CR,CR
```

Regulations

1. **Deadline:** The deadline for this homework is strict and no late submissions will be accepted. Submission deadlines are **not** subject to postponement.
2. **Programming Language:** Your code should be written in Python3. Your submission will be tested in inek machines. So make sure that it can be executed on them.
3. **Implementation:** Your code should not import any library other than **numpy** and **queue**.
4. **We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing or codes taken from internet included) will be punished according to the university regulations.
5. **Discussion:** You must follow OdtuClass for discussions and possible updates on a daily basis.
6. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. A reasonable timeout will be applied according to the complexity of test cases. This is not about the code efficiency, its only purpose is avoiding infinite loops due to an erroneous code.
7. **Submission:** Submission will be done via OdtuClass. You should upload a **single** python file named `<your_student_id>_hw1.py`, e.g., `e1234567_hw1.py`.